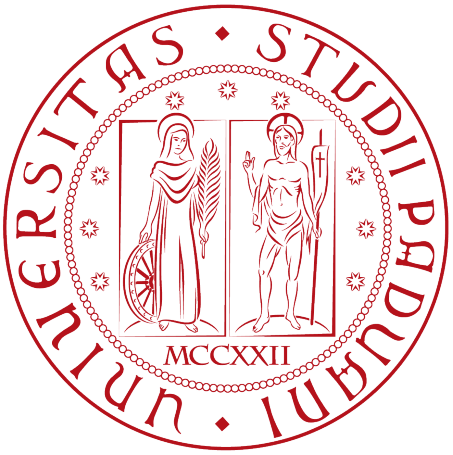
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Degree Course in Computer Science



A Practical Approach to Containerization & Distribution of Game Engines

Master degree thesis

Thesis supervisor

Prof. Claudio Enrico Palazzi

Co-advisors

Prof. Dario Maggiorini
Prof. Paolo Giaccone

Graduate student

Alessandro Sgreva

Academic Year 2022-2023

Summary

General description of the time and scope of the thesis, including a brief description of the outputs of the work.

With the aim of preventing possible ambiguity, the technical terms used in the present document are clarified and elaborated on in the *appendix A - Glossary.* Furthermore, in order to facilitate the reading of the document, said terms are marked with a subscript *'G'.*

Special thanks

First of all, I would like to express my gratitude to Prof. Claudio Enrico Palazzi, supervisor of my thesis, and to professors Paolo Giaccone and Dario Maggiorini, for their availability and support during the project experience, as well as during the drafting of the thesis document.

I would like to thank my mother for the help, support and patience brought during these years of university study.

Finally, I would also like to thank my friends and colleagues for the company, the support and the experiences lived together.

Padova, February 2023 Alessandro Sgreva

Contents

List of Figures

List of Tables

Chapter 1

Introduction

*This chapter introduces the concept of **Game Engine**, discusses the main problems of **Legacy Game Engines** with respect to **Distributed Game Engines**, and presents the main purpose of our research work.*

1.1 Background & Problems

Nowadays, *Game Engines* have become a core element in the development of most video game software [ref]. They are often identified as a suite of tools, that is able to provide core and important functionalities for managing basic features of video games, such as: image rendering, physics management, animation and many more [ref].
These type of features, while still being paramount for the execution of a video game, are often expensive to implement from scratch for each new software being developed. Furthermore, their nature and value can often be reused or adapted to different projects, making the development process much more efficient.

On a more technical level, the Game Engines are comprised of a tool suite and a run-time component [ref Gregory]. The tool suite allows creators to merge together various kinds of multimedia and audiovisual assets, while the run-time is a layered software that takes care of background operations (e.g. resource management, interaction with the hardware, ...) and is also transferred into the game executables, which as important consequences. In fact, even if multiple Game Engines present a certain degree of modularization, with clear separation of the provided functionalities in different modules, they still designed with a monolithic structure.

This type of Game Engine, which we call *Legacy Game Engine*, is presented with the strong requirement of high performance and low delays for the user interfacing with the image rendered on screen. This is particularly problematic in contexts where the available resources are limited or network latency is present.
Furthermore, due to their monolithic nature, even small changes can require extensive refactoring of their whole codebase and their interfacing with the hardware can introduce problematic platform dependency, which hinders portability.

In an effort to find a solution to these problems, research has been conducted on more modular and decentralized Game Engines [ref], which we will call *Distributed Game Engines*. This type of Game Engines can be implemented with many different

approaches [ref], but are generally hosted on multiple physical or virtual machines. In particular, this structure decouples the various GE modules (e.g. rendering, physics, IA, ...) and hosts them in different machines, which communicate with each other in order to provide the features of a full Game Engine. The aim of this type of architecture is to provide the user with a low delay full-functioning Game Engine, without the limitations of a monolithic structure and with more flexible resource allocation for the single functionalities.

This paradigm has proven to be quite successful, especially in the context of *Cloud Computing*, allowing the offloading of the most computation-intensive operations to dedicated hosting services, thus overcoming many limitations of local hardware. However, when distributing the components of a software into a network environment, additional elements such as network latency are introduced into the picture. As such, when designing technical solutions, it is important reason on the impact on performance of positive and negative side effects.

In this context, even if some previous works have defined the requirements for such architecture [ref], little research has been conducted on a practical implementation, considering its possible problems and technological tools to implement it.

1.2 Purpose & Research question

In this work, we verify the possibility of turning a Legacy Game Engine into a Distributed Game Engine, through modularization and containerization of its main components. We propose a Docker-based architecture where the Game Engine modules, such as: graphic engine, physics engine, AI, music player, ...; are connected and communicate in a network environment.
Additionally, we also introduce original functionalities fitting for a peer-to-peer distributed system, such as game image remote streaming and game state synchronization, through dedicated middlewares.

Considering the requirement of distributing Game Engine data between various virtual containers, we also compare possible distributed database solutions (e.g. ETCD, Redis), understanding their positive and negative aspects, with the aim of identifying the most fitting solution for our purpose.

In order to provide a more practical approach to the design of our architecture, we also consider realistic problems of distributed network environments (e.g. network latency, network traffic, synchronization, ...) and perform dedicated experiments to quantify their impact on the system performance and functionalities.

To sum up, the main research questions this work aims to answer are as follows:

* is it possible to decouple and containerize Game Engine modules or libraries, while maintaining the original system functionalities?
* what are the most fitting and performing options for distributing Game Engine data across multiple components in a network environment?
* what are the effects of network latency and traffic on the performance of a Distributed Game Engine?

Chapter 2

State of the art

This chapter discusses the state-of-the-art in relation to the scope this project, with a coverage of multiple topics which consitute the background of our work.

2.1 An overview on Game Engines

Discussion on the main features of Game Engines, with an overview of Legacy Game Engines and their shortcomings.

2.2 Distributed Game Engines

Discussion of the general design proposals in the context Distributed Game Engines, which are presented as a solution to the shortcomings of Legacy Game Engines.

2.2.1 Proposed architectures

Discussion of some proposed distributed architectures for Distributed Game Engines, including: client-server, cloud gaming, computation offloading.

2.2.2 Game responsiveness vs. User tolerance

Discussion of the problem of game responsiveness (including network latency) against the tolerance of players to delays and poor performance.

2.2.3 System scalability

Discussion of the problem of scalability, which introduces additional traffic (thus network latency) and synchronization issues.

2.3 Synchronization of a shared Game State

Discussion about the distribution and synchronization of a shared Game State between multiple Game Engines.

Chapter 3

Technologies

*This chapter discusses the technologies **used** or **considered** during the project, providing an in-depth description and motivating their choice.*

3.1 Docker

General description of Docker and Docker engine

3.1.1 Docker-compose

General description of Docker-compose and its features.

3.1.2 Nvidia-docker

General description of nvidia-docker, its features and incompatibility with Docker-compose.

3.2 ETCD

General description of ETCD, its features and possible implementations (Docker, APIs, ...).

3.3 Redis

General description of Redis, its features and possible implementations (Docker, APIs, ...).

Chapter 4

Feasibility study

This chapter discusses the scope of the thesis, with the feasibility study conducted on multiple softwares and introducing the main subject of experimentation, which is the ***TORCS game engine****.*

(The main purpose of this whole chapter is to provide the reader with a clear understanding of the reasons behind the choice of TORCS as game engine for our tests.)

4.1 Preliminary analysis

This section presents the general study conducted on various software options, in order to evaluate their fitness for the purpose of the project.

4.1.1 Godot

Description of Godot with pros and cons with respect to the purpose of the project.

4.1.2 Python Minecraft Clone

Description of the Python Minecraft Clone with pros and cons with respect to the purpose of the project.

4.1.3 Flightgear

Description of the Flightgear with pros and cons with respect to the purpose of the project.

4.1.4 TORCS - The Open Racing Car Simulator

Description of the Torcs with pros and cons with respect to the purpose of the project.

4.1.5 Fofix

Description of the Fofix with pros and cons with respect to the purpose of the project.

4.1.6 Conclusion

Summary and final decision (on TORCS).

4.2 Technical analysis

Techinical analysis of TORCS, considering its main components and state management. Will most likely include multiple subsections dedicated to the various aspects of the software.

4.2.1 Architecture of TORCS

In-depth description of the TORCS architecture.

4.2.2 Model View Controller in TORCS

*To provide additional reasoning about the software structure, we might include the discussion about the **MVC pattern** applied to TORCS.*

4.2.3 Simulated Car Racing Championship Competition (SCR)

In-depth description of the SCR client-server architecture, with the action-state communication.

Chapter 5

Software development

*This chapter discusses the development done on the initial TORCS codebase, from the starting point introduced in the previous chapter. The discussion includes the **containerization** of the main TORCS system components, the introduction of **additional software** (e.g. ETCD) and the development of **new components** (e.g. the state management middleware).*

5.1 Codebase definition

In this section we discuss some TORCS related projects found on GitHub, whose ideas provided an useful development basis or inspiration for our work.

5.1.1 Patched TORCS 1.3.7

*General description of this patched version of TORCS, which includes the **SCR client-server architecture** and the **screenpipe video streaming**. Here we can also discuss the reasons for choosing this project as starting point for the development of our work.*

5.1.2 PyTorcs-docker

General description of the PyTorcs-docker project, including its development of the original SCR client-server architecture, the usage of nvidia-docker and of the SnakeOil library. We also disccuss the reasons for not choosing this project as a starting point, which may be summed up as:

- *this project had an already clear development direction and removed multiple original TORCS functionalities for that purpose;*
- *this project acted as a Python wrapper of the original TORCS code, introducing a not-needed layer of complexity for our work;*
- *while providing interesting features, nvidia-docker is not compatible with docker-compose, which could impose techical limitations for our project development.*

5.2 Containerization of TORCS

*In this section we discuss the initial transition of TORCS from a local Game Engine, towards its implementation with Docker containers, starting from its patched version. This includes the **SCR client-server configuration** with two different containers and references to **X11** for the local display of the game image coming from the container.*

5.3 Implementation and improvement of ETCD

*In this section we discuss the **implementation** of ETCD as a Docker container, its **configurations** and the **improvements** introduced in order to adapt it for the purpose of the current project. Despite being triggered by the results of the experiments we performed, such experiments are only briefly referenced in this section, as their are discussed in-depth in later sections.*

5.4 Game image streaming

*In this section we discuss the initial implementation of the **screenpipe** game image streaming, the corrections introduced in order for it to work as expected and the improvements we introduced. These improvements will reference mainly the **introduction of ETCD** and the **refactoring**, which removed the need for IPC shared memory and the data serialization.*

5.5 ETCD state-action communication

*In this section we discuss the introduction of ETCD as an alternative to socket-based communication for the interaction between client and server in the **SCR configuration**. More specifically, we discuss how it was possible to allow for the exchange of **states and actions** between the client and the server using **ETCD keys**.*

5.6 Music Player library

*In this section we discuss the study behind the possibility of decoupling some of the TORCS libraries, which lead to the decision of focusing on the **Music Player library**, and how this operation was carried out.*

5.7 State Manager Middleware

*In this section we discuss the development of the State Manager Middleware, referencing the works and ideas mentioned in the **Related Works** section of chapter 1, and providing detailed technical information about the structure of the developed component.*

Chapter 6

Experimental methodology

This chapter discusses the single experiments conducted during the project, providing their ***purpose*** *and* ***expected theoretical outcome. Moreover, this chapter introduces the research methods and design that were used to conduct the studies, both for the qualitative and the quantitative experiments.***

6.1 Qualitative experiments

The following sections present qualitative experiments, performed in order to obtain a general idea about the effects on the system of specific development decisions. These experiments can provide general numerical data, which is generally ***not*** *supported by multiple misurations or graphical representations.*

6.1.1 ETCD for SCR state-action communication

This section presents the introduction of ETCD for SCR state-action communication, as a replacement for the socket-based original alternative.

Purpose & Expected outcome

Methods & Design

6.1.2 Multi-node ETCD cluster

This section presents the introduction of an ETCD cluster of 3 members, verifying the effect this had on the system performance.

Purpose & Expected outcome

Methods & Design

6.2 Quantitative experiments

The following sections present quantitative experiments, performed in order to obtain precise and numerical data about specific phenomena, effects on the system performance of specific configurations or correlation between system components. The data provided by these experiments is supported by multiple misurations and graphical representations.

6.2.1 X11 forwarding performance assessment

This section presents the test conducted in order to verify whether X11 can be considered a bottleneck and have significant impact on the performance of video streaming between two different PCs.

Purpose & Expected outcome

Methods & Design

6.2.2 Game image streaming solutions

This section presents the experiment performed in order to measure the video streaming performance of the system with different configurations.

Purpose & Expected outcome

Methods & Design

6.2.3 Network traffic analysis

This section presents the experiment performed in order to measure the amount of data processed in I/O from/towards the network or the disk, alongside measuring the round-trip-time between SCR client and server in the same architecture.

Purpose & Expected outcome

Methods & Design

6.2.4 Network latency impact assessment

This section presents the experiment performed in order to measure the performance of TORCS and screenpipe displays, in situations with high or moderate network latency in the ETCD container.

Purpose & Expected outcome

Methods & Design

6.2.5 Distribution of dynamic game state data

This section presents the experiment performed in order to measure the system performance when introducing the storage of an increasing number of game state data fields into ETCD, alongside an increasing amount of network latency in the ETCD container.

Purpose & Expected outcome

Methods & Design

6.2.6 Distribution of static game state data

This section presents the experiment performed in order to measure the system performance when introducing the storage of different static game state data fields into ETCD.

Purpose & Expected outcome

Methods & Design

6.2.7 ETCD in-memory persistence

This section presents the experiment performed in order to measure the system performance when storing 3 game state data fields in a situation in which ETCD is writing to memory, instead of the Disk.

Purpose & Expected outcome

Methods & Design

6.2.8 Graphics and physics engine correlation

This section presents the experiment performed in order to verify and quantify the correlation between the graphic and physics engine of TORCS, by introducing an increasing delay into the simulation module and verifying the impact on the graphics framerate. Moreover, we also compute the theoretical framerate bounds generated by the delays and compare them with the actual measured framerate.

Purpose & Expected outcome

Methods & Design

6.2.9 Graphics and game engine framerate correlation

This section presents the experiment performed in order to measure the net operational time and the correlation between the GE framerate and the graphics framerate, in a situation in an increasing delay introduced into the simulation module.

Purpose & Expected outcome

Methods & Design

Chapter 7

Experiments results & discussion

*In this chapter provides the results of the previously described experiments, with clear representation of data and highlighting the most relevant elements. These results are discussed then in order to provide a correlation with respect to the **theoretical expectations** and the **research questions**.*

7.1 Qualitative experiments

*The following sections present qualitative experiments, performed in order to obtain a general idea about the effects on the system of specific development decisions. These experiments can provide general numerical data, which is generally **not** supported by multiple misurations or graphical representations.*

7.1.1 ETCD for SCR state-action communication

This section presents and discusses the results obtained by the introduction of ETCD for SCR state-action communication, as a replacement for the socket-based original alternative.

7.1.2 Multi-node ETCD cluster

This section presents and discusses the results obtained by the introduction of an ETCD cluster of 3 members, verifying the effect this had on the system performance.

7.2 Quantitative experiments

The following sections present quantitative experiments, performed in order to obtain precise and numerical data about specific phenomena, effects on the system performance of specific configurations or correlation between system components. The data provided by these experiments is supported by multiple misurations and graphical representations. As such, the structure of the sections may vary depending on the specific experiment.

7.2.1 X11 forwarding performance assessment

This section presents and discusses the results obtained by test conducted in order to verify whether X11 can be considered a bottleneck and have significant impact on the performance of video streaming between two different PCs.

7.2.2 Game image streaming solutions

This section presents and discusses the results obtained by the experiment performed in order to measure the video streaming performance of the system with different configurations.

7.2.3 Network traffic analysis

This section presents and discusses the results obtained by the experiment performed in order to measure the amount of data processed in I/O from/towards the network or the disk, alongside measuring the round-trip-time between SCR client and server in the same architecture.

7.2.4 Network latency impact assessment

This section presents and discusses the results obtained by the experiment performed in order to measure the performance of TORCS and screenpipe displays, in situations with high or moderate network latency in the ETCD container.

7.2.5 Distribution of dynamic game state data

This section presents and discusses the results obtained by the experiment performed in order to measure the system performance when introducing the storage of an increasing number of game state data fields into ETCD, alongside an increasing amount of network latency in the ETCD container.

7.2.6 Distribution of static game state data

This section presents and discusses the results obtained by the experiment performed in order to measure the system performance when introducing the storage of different static game state data fields into ETCD.

7.2.7 ETCD in-memory persistence

This section presents and discusses the results obtained by the experiment performed in order to measure the system performance when storing 3 game state data fields in a situation in which ETCD is writing to memory, instead of the Disk.

7.2.8 Graphics and physics engine correlation

This section presents and discusses the results obtained by the experiment performed in order to verify and quantify the correlation between the graphic and physics engine of TORCS, by introducing an increasing delay into the simulation module and verifying the impact on the graphics framerate. Moreover, we also compute the theoretical framerate bounds generated by the delays and compare them with the actual measured framerate.

7.2.9 Graphics and game engine framerate correlation

This section presents and discusses the results obtained by the experiment performed in order to measure the net operational time and the correlation between the GE framerate and the graphics framerate, in a situation in an increasing delay introduced into the simulation module.

Chapter 8

Conclusion

*This chapter provides a final discussion about the outcome of the whole research effort, considering also the **limitations** of the study design/findings, and the **recommandations for future research**.*

8.1 Final assessment

This section discusses the final assessment on the whole project, summing up the development and the results obtained by the experiments, to provide answers to the initially presented research questions and problem.

8.2 Limitations

This section discusses the limitations of the study design/findings, providing some additional context and scope for the previous assessment.

8.3 Recommandations for future research

This section provides some useful and insightful ideas for future works. This could be an ideal location for referencing alternatives such as REDIS and OrbitDB, or even all the ones mentioned in Antonio's study.

Appendix A

Glossary

.NET: framework per l'esecuzione di applicazioni ad esso destinate. È costituito da Common Language Runtime, che fornisce la gestione della memoria ed altri servizi di sistema, e da un'ampia libreria di classi.

UML: linguaggio di modellazione e di specifica basato sul paradigma orientato agli oggetti.

Bibliografy

Bibliographical references

[1] Abdulbaset Abdulaziz Gaddah. *A Pro-Active Mobility Management Scheme for Publish/Subscribe Middleware Systems*. Umm Al-Qura University, 2015.

Web references

[1] *Sito ufficiale Euronovate*. URL: https://www.euronovate.com/.